OPENDIR

Vulnerable to TOCTOU issues

Sean Barnum, Cigital, Inc. [vita[1]]

Copyright © 2007 Cigital, Inc.

2007-04-02

Part "Original Cigital Coding Rule in XML"

Mime-type: text/xml, size: 8803 bytes

Attack Category	• Path spoofing or confusion problem
Vulnerability Category	• Indeterminate File/Path • TOCTOU - Time of Check, Time of Use
Software Context	• File Creation • File I/O
Location	• dirent.h
Description	The opendir() function opens a directory stream corresponding to the directory name and returns a pointer to the directory stream. The stream is positioned at the first entry in the directory. opendir() is vulnerable to TOCTOU attacks. A call to opendir() should be flagged if the argument (the directory name) is used previously in a check-category call.
APIs	**Function Name** **Comments** opendir use
Method of Attack	The key issue with respect to TOCTOU vulnerabilities is that programs make assumptions about atomicity of actions. It is assumed that checking the state or identity of a targeted resource followed by an action on that resource is all one action. In reality, there is a period of time between the check and the use that allows either an attacker to intentionally or another interleaved process or thread to unintentionally change the state of the targeted resource and yield unexpected and undesired results. The opendir() call is a use-category call, which when preceded by a check-category call can be indicative of a TOCTOU vulnerability. A TOCTOU attack in regards to opendir() can occur, for example, when a. A check for the existence of a directory occurs b. opendir() is called

1.   http://buildsecurityin.us-cert.gov/bsi-rules/35-BSI.html (Barnum, Sean)

	Between a and b, an attacker could, for example, link the target directory (the one to be opened) to a different known directory. The subsequent opendir() call would either fail or have unexpected results or behavior.
Exception Criteria	

Solutions	**Solution Applicability**	**Solution Description**	**Solution Efficacy**
	Generally applicable to all opendir() calls.	Utilize a file descriptor version of check and use functions.	Effective.
	Generally applicable to all opendir() calls.	The most basic advice for TOCTOU vulnerabilities is to not perform a check before the use. This does not resolve the underlying issue of the execution of a function on a resource whose state and identity cannot be assured, but it does help to limit the false sense of security given by the check.	Does not resolve the underlying vulnerability but limits the false sense of security given by the check.
	Generally applicable to all opendir() calls.	Limit the interleaving of operations on files from multiple processes.	Does not eliminate the underlying vulnerability but can help make it more difficult to exploit.
	Generally applicable to all opendir() calls.	Limit the spread of time (cycles) between the check and use of a resource.	Does not eliminate the underlying vulnerability but can help make it more

			difficult to exploit.
	Generally applicable to all opendir() calls.	Recheck the resource after the use call to verify that the action was taken appropriately.	Effective in some cases.

Signature Details	DIR *opendir(const char *name);
Examples of Incorrect Code	(see code below)

```
/* check has been added */
#include "dirent.h"
#include "errno.h"
#include "sys/stat.h"
#include "sys/types.h"
#include "stdio.h"

void traverse(char *fn, int
indent) {
DIR *dir;
struct dirent *entry;
int count;
char path[1025];
struct stat info;
int check_status;
struct stat statbuf;

for (count=0; count<indent; count+
+) printf(" ");
printf("%s\n", fn);

check_status=stat(fn, &statbuf);
...

if ((dir = opendir(fn)) == NULL)
perror("opendir() error");
else {
while ((entry = readdir(dir)) !=
NULL) {
if (entry->d_name[0] != '.') {
strcpy(path, fn);
strcat(path, "/");
strcat(path, entry->d_name);
if (stat(path, &info) != 0)
fprintf(stderr, "stat() error on
%s: %s\n", path,
strerror(errno));
else if (S_ISDIR(info.st_mode))
traverse(path, indent+1);
}
}
closedir(dir);
}
}
```

| | ```
main() {
puts("Directory structure:");
traverse("/dev", 0);
}
``` |

| **Examples of Corrected Code** | ```c
/* This is a somewhat better
solution, but it ties off the
ability to do any "checks" */
/* prior to usage of the
opendir(). */

#include <dirent.h>
#include <errno.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <stdio.h>

void traverse(char *fn, int
indent) {
DIR *dir;
struct dirent *entry;
int count;
char path[1025];
struct stat info;

for (count=0; count<indent; count+
+) printf(" ");
printf("%s\n", fn);

if ((dir = opendir(fn)) == NULL)
perror("opendir() error");
else {
while ((entry = readdir(dir)) !=
NULL) {
if (entry->d_name[0] != '.') {
strcpy(path, fn);
strcat(path, "/");
strcat(path, entry->d_name);
if (stat(path, &info) != 0)
fprintf(stderr, "stat() error on
%s: %s\n", path,
strerror(errno));
else if (S_ISDIR(info.st_mode))
traverse(path, indent+1);
}
}
closedir(dir);
}
}

main() {
puts("Directory structure:");
traverse("/dev", 0);
}
``` |
| **Source References** | • Viega, John & McGraw, Gary. *Building Secure Software: How to Avoid Security Problems* |

	the Right Way. Boston, MA: Addison-Wesley Professional, 2001, ISBN: 020172152X, ch 9 • UNIX man page for opendir()	
Recommended Resource		
Discriminant Set	**Operating System**	• UNIX
	Languages	• C
		• C++

Cigital, Inc. Copyright

---

1.   mailto:copyright@cigital.com